1. **Дайте определение массиву.**

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку).

**1.1. Как осуществляется индексация элементов массива**.

Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют **индексом**.

В случае с Java массив однороден, то есть во всех его ячейках будут храниться элементы одного типа. Так, массив целых чисел содержит только целые числа (например, типа int), массив строк — только строки, массив из элементов созданного нами класса Dog будет содержать только объекты Dog. То есть в Java мы не можем поместить в первую ячейку массива целое число, во вторую String, а в третью — “собаку”.

**1.2. Как необходимо обращаться к i-му элементу массива?**

Доступ к конкретному элементу массива осуществляется по индексу (индексам).

1. **Приведите способы объявления и инициализации одномерных и двумерных массивов примитивных и ссылочных типов.**

**Первый способ(**предпочтительный способ**):** dataType[] arrayName;

**int**[] myArray;

Object[]arrayOfObjects;

int[] name = new int [5];

int[][] name = new int [4][5]; где – 4 строки и 5 – столбцов в массиве.

**Второй способ**: dataType arrayName[];

**int** myArray[];

int name[] = new int [5];

int name[] = {4,6,8,98,25};

Object arrayOfObjects[];

int name [][] = new int [4][5];

После создания массива с помощью new, в его ячейках записаны значения по умолчанию. Для численных типов это будет 0, для

boolean — false, для ссылочных типов — null.

2.1. **Укажите разницу, между массивами примитивных и ссылочных типов.**

Примитивный тип резервируется по умолчанию нулевыми значениями.

Ссылочный тип –null.

Массив- это ссылочный тип.

1. **Объясните, что значит клонирование массива, как в Java можно клонировать массив, в чем состоит разница в клонировании массивов примитивных и ссылочных типов.**

**При клонировании ссылочного типа массива нужно скопировать все вложенные ссылки на объект.**

## Методы для копирования массивов в Java

В Java существует довольно много специальных методов для копирования массивов:

1.В первую очередь, хотелось бы упомянуть Object.clone() — этот метод вы можете использовать для полного копирования массива. Соответственно, если вы хотите скопировать массив частично, этот способ вам не подойдёт.

2.Следующий на очереди — System.arraycopy() — по сути, это один из наилучших способов создать частичную копию массива в Java. В этом методе определены следующие параметры: — массив, элементы которого планируем копировать; — индекс элемента; — итоговый (результирующий) массив; — первый элемент итогового массива; — общее число элементов, предназначенных для копирования.

К примеру, написав **System.arraycopy(источник, 2, назначения, 5, 7)**, вы скопируете семь элементов из массива-источника в итоговый массив, начиная со второго индекса источника в пятый индекс результирующего массива.

3.Arrays.copyOf() — подойдёт вам, если планируете выполнить копирование нескольких первых элементов массива либо сделать полную копию массива. Способ не так универсален, как System.arraycopy(), но так же прост в применении.

4.Arrays.copyOfRange() — полезный метод, обеспечивающий частичное копирование массива.

В принципе, для решения большинства задач по полному либо частичному копированию массивов в Java вышеперечисленных методов вам вполне хватит. Только учтите, что методы, встроенные в Java для копирования, годятся лишь для одномерных массивов.

1. **Объясните, что представляет собой двумерных массив в Java, что такое “рваный массив”. Как узнать количество строк и количество элементов в каждой строке для “рваного” массива?**

**Двумерный массив в Java** — это **массив массивов**, то есть в каждой его ячейке находится ссылка на некий **массив**. Но гораздо проще его представить в виде таблицы, у которой задано количество строк (первое измерение) и количество столбцов (второе измерение).

Зубчатый массив в Java - это многомерный массив, содержащий в качестве элементов **массивы различных размеров**. Он также называется «**массивом массивов**» или «**рваным массивом**».

Простой **способ определить зубчатый массив**: int[][]jaggedArr = {{1, 2}, {3, 4, 5}, {6, 7, 8, 9}};

**Пример:**

int[][]jaggedArr = new int[3][]; Здесь **мы не указали второе измерение, поскольку оно будет варьироваться**.  Дальше, объявив и инициализировав соответствующие элементы в jaggedArr :

jaggedArr[0]= new int[]{1, 2};

jaggedArr[1]= new int[]{3, 4, 5};

jaggedArr[2]= new int[]{6, 7, 8, 9};

Также можем просто объявить его элементы без их инициализации:

jaggedArr[0]= new int[2];

jaggedArr[1]= new int[3];

jaggedArr[2]= new int[4]; Затем они могут быть позже инициализированы, например, с использованием пользовательских данных.

1. **Объясните ситуации, когда в java-коде могут возникнуть следующие исключительные ситуации java.lang.ArrayIndexOutOfBoundsException и java.lang.ArrayStoreException.**

Когда мы вышли за границы нашего массива.

Пример:

public class Main {

public static void main(String[] args){

int[]arr = {5,7,2,8,1,0};

for (int i=0;i<=arr.length;i++){

int min = arr[i];

int mini = i;

for(int i2=i;i<=arr.length;i2++){

if (arr[i2]<min){

min=arr[i2];

mini=i2;

}

}

arr[mini] = arr[i];

arr[i] = min;

}

System.out.println(Arrays.toString(arr) );

}

}

**Ошибка находится в if (arr[i2]<min). У нас 6 элементов в int[]arr = {5,7,2,8,1,0};, нумерация элементов идет с 0-го, то есть последний элемент будет под индексом 5. arr.length вернет длину массива, а она равна 6-ти. В циклах for при проходе на последнем шаге пытались обратиться в элементу под индексом 6 (arr[arr.length] - 6-й элемент), а его нет. Поэтому ставим строго < в цикле при проверке условия.**

Исправленный код

int[]arr = {5,7,2,8,1,0};

**for (int i = 0; i < arr.length; i++){ // ВОТ ТУТ**

int min = arr[i];

int mini = i;

for (int i2 = i; i2 < arr.length;i2++){

if (arr[i2]<min){

min=arr[i2];

mini=i2;

}

}

arr[mini] = arr[i];

arr[i] = min;

}

System.out.println(Arrays.toString(arr) );

**java.lang.ArrayStoreException.**

Означает, что была сделана попытка сохранить объект неправильного типа в массиве объектов; (программа попыталась сохранить в массив значение неправильного типа).

Такая попытка становится возможно из-за [ковариантности](https://itsobes.ru/JavaSobes/chto-takoe-kovariantnost-i-kontravariantnost/) массивов.  
  
Ковариантность позволяет работать с массивом по типу массива родителей. Например, через приведение к  **Object[]** можно попытаться положить любой объект в любой массив:

Object x[] = new String[3];  
 x[0] = new Integer(0);

Компилятор гарантирует, что когда берем элемент из массива, он будет представителем типа элементов самого этого массива. Не важно какого типа переменная его хранит. Именно для обеспечения этой гарантии работает проверка типа *времени выполнения*, которая и выбрасывает **ArrayStoreException.**

1. **Объясните, зачем при кодировании разделять решаемую задачу на методы. Поясните, как вы понимаете выражение: “Один метод не должен выполнять две задачи”**.

Метод — это совокупность команд, позволяющая выполнить некоторую операцию в программе. Иными словами, метод — это некоторая функция; что-то, что умеет делать твой класс.

Для того, чтобы каждый метод выполнял строго свою функцию, было удобно разбираться в коде, не было путаницы, каждый метод выполняет строго свою задачу (функцию; действие).

Метод не должен выполнять две задачи – т.е.метод должен выполнить строго одну свою задачу, действие.

1. **Объясните, как в Java передаются параметры в методы, в чем особенность передачи в метод значения примитивного типа, а в чем ссылочного.**

**Java передает параметры по значению. Всегда.** Это означает — "скопировать значение и передать копию."

int х = 5;

doStuff (х); / / Передать копию х (значение 5) в метод doStuff. Копия значения **x**, тоесть 5, передается в метод **doStuff ()**.

При ссылочном - копируется значение ссылки.

**8. Объясните, как в метод передать массив.**

public static void printArray(int[] array) {

for (int i = 0; i < array.length; i++) {

System.out.print(array[i] + " ");

}

}

Его можно вызвать путем передачи массива. Например, следующий оператор вызывает метод printArray для отображения 3, 1, 2, 6, 4 и 2:

printArray(new int[]{3, 1, 2, 6, 4, 2});

**8.1. И как массив вернуть из метода.**

Метод может также возвращать массив. Например, метод, показанный ниже, возвращает массив, который является реверсирование другого массива:

public static int[] reverse(int[] list) {

int[] result = new int[list.length];

for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {

result[j] = list[i];

}

return result;

}

**8.2**.**Можно ли в методе изменить размер переданного массива.**

**Нельзя изменить размер массива при передаче в метод. Так как под каждый тип массива выделяется определенный размер памяти**, тип **int** имеет размер 4 **байта** (32 бита) и мы не можем например засунуть long 8 байт 64 бита.

**9. Поясните, что означает выражение ‘вернуть значение из метода’.**

Метод возвращается к коду, который вызвал его когда он

* завершает все операторы в методе,
* достигает **return** оператор, или
* выдает исключение.

Тип данных возвращаемого значения должен соответствовать объявленный тип возврата метода; невозможно возвратить целочисленное значение из метода, который, как объявляют, возвратил булево.

**9.1**. **Как можно вернуть значение из метода**.

**Методы** могут возвращать некоторое **значение**. Для этого применяется оператор return. return возвращаемое\_значение; После оператора return указывается возвращаемое **значение**, которое является результатом **метода**.

**9.2**. **Есть ли разница при возврате значений примитивного и ссылочного типов.**

Примитивные возвращают копию значения, а ссылочные копию ссылки.

**10. Перечислите известные вам алгоритмы сортировки значений, приведите код, реализующий это алгоритмы.**

Простейшая сортировка (Bubble Sort)

Сортировка выбором (Selection Sort)

Сортировка вставками (Insertion Sort)

Челночная сортировка (Shuttle Sort)

Сортировка Шелла

Cортировка слиянием (merge sort)

Сортировка подсчётом (Counting Sort)

Поразрядная сортировка (Radix Sort)

Быстрая сортировка (Quick Sort**)**

**Код сортировок будет реализован в Github.**

//задаём шахматную доску двумерным массивом

String [][] chessBoard = **new** String[8][8];

**for** (**int** i = 0; i< chessBoard.length; i++) {

**for** (**int** j = 0; j < chessBoard[0].length; j++) {

**if** ((i + j) % 2 == 0) chessBoard[i][j] = "W";

**else** chessBoard[i][j] = "B";

}

}

второй способ

String [][] chessBoard = **new** String[8][8];

**for** (**int** i = 0; i < chessBoard.length; i++) {

**for** (**int** j = 0; j < chessBoard[0].length; j++) {

**if** ((i + j) % 2 == 0) chessBoard[i][j] = "W" + chessBoardCoord(j,i);

**else** chessBoard[i][j] = "B"+ chessBoardCoord(j,i);

}

}

**for** (**int** i = 0; i < chessBoard.length; i++) {

**for** (**int** j = 0; j < chessBoard[0].length; j++) {

System.out.print(" " + chessBoard[i][j] + " ");

}

System.out.println();

}

//объявляем и создаём массив, указывая только количество строк

**int** [][] twoDimArray = **new** **int**[5][];

//инициализируем массив, заполняя его массивами разной длины

twoDimArray[0] = **new** **int**[]{1, 2, 3, 4, 5};

twoDimArray[1] = **new** **int**[]{1,2,3,4};

twoDimArray[2] = **new** **int**[]{1,2,3};

twoDimArray[3] = **new** **int**[]{1,2};

twoDimArray[4] = **new** **int**[]{1};

//выведем получившийся непрямоугольный двумерный массив на экран

**for** (**int** i = 0; i < twoDimArray.length; i++) {

**for** (**int** j = 0; j < twoDimArray[i].length; j++) {

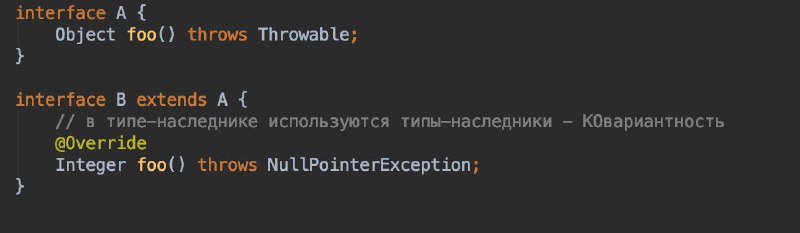
System.out.print(" " + twoDimArray[i][j] + " ");

}

System.out.println();

}

[Формально](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%B0%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), ковариантность/контравариантность типов – это сохранение/обращение порядка наследования для производных типов. Проще говоря, когда у ковариантных сущностей типами-параметрами являются родитель и наследник, они сами становятся как бы родителем и наследником. Контравариантные наоборот, становятся наследником и родителем.  
  
Легче всего осознать эти понятия на примерах:  
**🔘** **Ковариантность**: List<Integer> можно присвоить в переменную типа List<? extends Number> (как будто он наследник List<Number>).  
**🔘** **Контравариантность**: в качестве параметра метода List<Number>#sort типа Comparator<? super Number> может быть передан Comparator<Object> (как будто он родитель Comparator<Number>)  
  
Отношение типов «можно присвоить» – не совсем наследование, такие типы называются *совместимыми* (отношение «is a»).  
  
Массивы ковариантны: в переменную Object[] можно присвоить значение типа String[].  
  
Переопределение методов начиная с Java 5 ковариантно относительно типа результата и типов исключений.

[](https://itsobes.ru/assets/JavaSobes/62.jpg)

| **Примитивный тип** | **Размер в памяти** | **Диапазон значений** |
| --- | --- | --- |
| byte | 8 бит | от -128 до 127 |
| short | 16 бит | до -32768 до 32767 |
| char | 16 бит | от 0 до 65536 |
| int | 32 бита | от -2147483648 до 2147483647 |
| long | 64 бита | от -9223372036854775808 до 9223372036854775807 |
| float | 32 бита | от (2 в степени -149) до ((2-2 в степени -23)\*2 в степени 127) |
| double | 64 бита | от (-2 в степени 63) до ((2 в степени 63) - 1) |
| boolean | 8 (при использовании в массивах), 32 (при использовании не в массивах) | true или false |